Bash-scripting

Variables

```
# There can't be any space between the variable name and the equal sign. It has to be varname=command
battery_time=$(cat /sys/class/power_supply/BAT0/capacity)

# The variables can then be used like this
echo "$battery_time"
```

Iterate over a file

This script will iterate over a file and echo out every single line:

```
#!/bin/bash

for line in $(cat file.txt);do
    echo $line
done
```

Another way of writing is this:

```
#!/bin/bash

while read p; do
    echo  $p
done <file.txt
```

For-loops

```
#!/bin/bash

for ((i = 0; i < 10; i++)); do
    echo $i
done
```

Another way to write this is by using the program `seq`. Seq is pretty much like `range()` in python. So it can be used like this:

```
#!/bin/bash

for x in `seq 1 100`; do
    echo $x
done
```

If statement

`$1` here represent the first argument.

```
if [ "$1" == "" ]; then
    echo "This happens"
fi
```

If/Else

```
#!/bin/bash

if [ "$1" == "" ]; then
    echo "This happens"
else
    echo "Something else happens"
fi
```

Functions

```
#!/bin/bash

function myfunction {
echo "hello world"
}
```

Command line arguments

Command line arguments are represented like this

```
#!/bin/bash

$1
```

This is the first command line argument.

Daemonize an execution

If you do a ping-sweep with host the command will take about a second to complete. And if you run that against 255 hosts I will take a long time to complete. To avoid this we can just deamonize every execution to make it faster. We use the `&` to daemonize it.

```
#!/bin/bash

for ip in $(cat ips.txt); do
    ping -c 1 $ip &
done
```

Use the output of command

It has happened to me several times that I want to input the output of a command into a new command, for example:

I search for a file, find three, and take the last line, which is a path. Now I want to cat that path:

```
#!/bin/bash

locate 646.c | tail -n 1
```

This can be done like this:

```
#!/bin/bash

cat $(locate 646.c | tail -n 1)
```